DIFFERENCE EXTRACTION BETWEEN TWO VERSIONS OF DATA-TABLES CONTAINING INTRA-REFERENCES

FIELD OF THE INVENTION

This invention relates generally to updating computer programs.

BACKGROUND OF THE INVENTION

With the ever increasing use of remote communication and in particular the Internet, new applications have been introduced such as commercial trade over the Internet, electronic supermarkets, distribution of computer products over the Internet, and others.

Both the popularity and availability of access to the Internet for common users have encouraged not only the distribution of products, but also the upgrade and update of the product under question from a remote site, using, to this end, the Internet infra-structure.

Turning to a specific example of computer programs, an *old program* is installed at a remote client site and is subject to be upgraded to a *new program,* where the latter includes some modifications as compared to the old program.

In order to carry out the update at the remote client site (through the network), the provider should, preferably, generate a difference result representative of the difference between the *old program* and the *new program,* and send the resulting file through the Internet to the remote client site. The client, in turn, invokes appropriate utility, which incorporates the differences in the old program, thereby generating the desired new program at the client site. The specified procedure carries the obvious advantages in that on the one hand, the provider does not need to be present at the client site and, on the other hand, only

the difference result and not the entire new program is sent to the client. Assuming, for example, that a modified *Office '97* package (commercially available from Microsoft Inc. USA) should be sent to clients, since the compressed size of programs of the package occupies tens of Mega-bytes, and, further considering the relatively low throughput of the Internet and the bottleneck of the modem throughput at the client end (say an average of 33,600 bps), it is easy to understand that transmitting the entire new package through the network is practically infeasible.

Normally, the volume of the difference result is significantly smaller than that of the raw new program and, accordingly, sending only the difference result data rather than the entire new program, is more efficient. This notwithstanding, and as will be explained in greater detail below, applying known *per se* file difference applications (such as techniques utilized by *diff* utilities of the UNIX operating systems or a similar *diff* utility of the GNU project from FSF) in order to generate a difference result between the old program and the new program, normally results in a relatively large amount of data, even if the modifications that were introduced to the old program (in order to generate the new program) are very few. Thus, consider, for example, an old program where few new instructions are inserted and few others are deleted in order to bring about the new program. The difference result between the old program and the new program will not only reveal the inserted and deleted instructions, but also all those entries that jump, jump on condition, call functions, reference to data and possibly others (referred to, collectively, as reference entries – see glossary below) which, by nature, specify a target address (reference) as an integral part of the command. The latter addresses may have been changed due to the fact that some instructions were added and others deleted. It is important to note that the reference entries that are modified are not those that were inserted, and obviously not those that were deleted. In fact, insertion of only one new entry may result in the plurality of altered reference entries which will naturally be reflected in the difference result and obviously will inflate its volume.

It is accordingly appreciated that despite the fact that the actual change between the old and new program is very limited, the resulting file difference is relatively large. The same problem is encountered in other applications, which employ data tables (see Glossary below), that are structured like a program and are subject to updates in the manner specified.

There is accordingly a need in the art to provide for an efficient tool which will result in significantly smaller volumes of difference results between old programs and new programs, as compared to hitherto known techniques for accomplishing difference result. The proposed tool is useful for various applications including, but not limited to, incremental software updates and version control.

There is yet another need in the art to provide for an efficient tool which will result in significantly smaller volumes of difference results between old data tables and new data tables.

GLOSSARY:

There follows a glossary of terms, some of which are conventional and others have been coined:

Data Table - a table of *entries*, each may have a different size;

Entry - a *data table* includes *entries*, each of which is an addressable unit that contains data;

Address - a number which is uniquely assigned to a single *entry* by which that *entry* is accessed; In the following description, the terms *entry* and *address* are occasionally used interchangeably.

Reference- a part of the data appearing in an *entry* in the *data table* which is used to refer to some other *entry* from the same *data table*. A *reference* can be either an *address* or a number used to compute an *address*. *Entries* that include *references* are designated also as *reference entries*.

Label - an abstract notation of an *entry* which is referred by another *entry* of the same *data table* through a *reference*.

Old Data Table - a data table (or portion of a data table) that is to be updated (possibly from remote site) so as to generate a *new data table* (or portion of a *new data table*). Insofar as remote update is concerned, it is normally, although not necessarily, transmitted through a communication network such as the Internet. It should be noted that whilst for convenience of explanation only, the description focuses predominately on the Internet, the invention is by no means bound by this specific example.

As an example, a *data table* can be an executable program either as a loaded program in machine-memory or as an executable-file. In this example, *entries* are individual machine instructions of the program or the individual data elements used by the program.

Instructions and data elements of a program may contain addresses to other instructions or data elements and are regarded as *references*. Such *references* can be detected by a process of disassembly applied on the program or, if given, by analyzing a relocation table attached to executable programs by link-editors that create them.

Another example of a *data table* is a group of inter-linked data records stored in an array of bytes where records contain addresses of other data records. The format of the records and the way they are laid out in the array are known, and the analysis and decomposition of such array is possible.

Old program – an example of old *data table*: a program (or portion of a program) that is to be updated so as to generate a *new program* (or portion of a program).

It should be further noted that reference to the *old program* and the *new program* is made for convenience of explanation only, and encompasses *inter alia* the upgrade of the old program to the new program (e.g. due to an upgrade in versions), modifications of the old program to the new program, (e.g. due to corrections of bugs in the old program), and changing from a first old program to a second (and possibly different) new program;

SUMMARY OF THE INVENTION

For convenience of explanation, the invention is described with reference to a specific example of computer programs. The invention is by no means bound by this particular example.

As explained above, applying a known *per se* file difference utility to an old program and a new program normally results in a relatively large amount of data, even if the modifications that were introduced to the old program (in order to generate the new program) are very few. The present invention is based on the observation that the relatively large size of the difference result stems from the alterations of reference in reference entries as a result of other newly inserted entries (and/or entries that were deleted).

On the basis of this observation, the invention aims at generating a modified old program and a modified new program, wherein the difference in references in corresponding entries in said new and old programs as explained above, will be reflected as invariant entries in the modified old and new programs. The net effect is that the invariant reference entries (between the modified old program and the modified new program), will not appear in the difference result, thereby reducing its size as compared to a conventional difference result obtained by using hitherto known techniques.

Accordingly, the invention provides for a method for generating a compact difference result between an old program and a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old program is generated;

(b) scanning the new program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new program is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old program and modified new program.

The invention further provides for a method for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old

program is generated;

(c) reconstituting the modified new program utilizing at least said compact difference result and said modified old program; said modified new program is differed from said new program at least in that substantially each reference entry in said new program is replaced in said modified new program by a distinct label mark;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

Still further, the invention provides for a method for generating a compact difference result between an old program and a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) generating a modified old program utilizing at least said old program;

(b) generating a modified new program utilizing at least said new program, said modified old program and modified new program have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/insert modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(c) generating said compact difference result utilizing at least said modified new program and modified old program.

Yet further, the invention provides for a method for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) generating a modified old program utilizing at least said old program;

(c) reconstituting a modified new program utilizing directly or indirectly at least said modified old program and said compact difference result; said modified old program and modified new program have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/inset modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

According to another aspect, the invention provides for a system for generating a compact difference result between an old program and a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of:

(a) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old program is generated;

(b) scanning the new program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new program is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old program and modified new program.

Still further, the invention provides for a system for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of :

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) scanning the old program and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old

program is generated;

(c) reconstituting the modified new program utilizing at least said compact difference result and said modified old program; said modified new program is differed from said new program at least in that substantially each reference entry in said new program is replaced in said modified new program by a distinct label mark;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

The invention further provides for a system for generating a compact difference result between an old program and a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of :

(a) generating a modified old program utilizing at least said old program;

(b) generating a modified new program utilizing at least said new program, said modified old program and modified new program have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/insert modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(c) generating said compact difference result utilizing at least said modified new program and modified old program.

Still further, the invention provides for a system for performing an update in an old program so as to generate a new program; each program including reference entries that contain reference that refer to other entries in the program; the system comprising a processing device capable of :

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old program and a modified new program;

(b) generating a modified old program utilizing at least said old program;

(c) reconstituting a modified new program utilizing directly or indirectly at least said modified old program and said compact difference result; said modified old program and modified new program have at least the following characteristics:

(i) substantially each reference in an entry in said old program that is different than corresponding entry in said new program due to delete/inset modifications that form part of the transition between said old program and new program are reflected as invariant references in the corresponding entries in said modified old and modified new programs;

(d) reconstituting said new program utilizing directly or indirectly at least said compact difference result and said modified new program.

Yet further, the invention provides for a method for generating a compact difference result between an old data table and a new data table; each data table

including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old data table is generated;

(b) scanning the new data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new data table is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old data table and modified new data table.

Moreover, the invention provides for a method for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old data table is generated;

(c) reconstituting the modified new data table utilizing at least

said compact difference result and said modified old data table; said modified new data table is differed from said new data table at least in that substantially each reference entry in said new data table is replaced in said modified new data table by a distinct label mark;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

The invention further provides for a method for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) generating a modified old data table utilizing at least said old data table;

(b) generating a modified new data table utilizing at least said new data table, said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/insert modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(c) generating said compact difference result utilizing at least said modified new data table and modified old data table.

The invention provides for a method for performing an update in an old data table so as to generate a new data table; each data table including reference

entries that contain reference that refer to other entries in the data table; the method comprising the steps of:

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) generating a modified old data table utilizing at least said old data table;

(c) reconstituting a modified new data table utilizing directly or indirectly at least said modified old data table and said compact difference result; said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/inset modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

The invention further provides for a system for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of:

(a) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified old data table is generated;

(b) scanning the new data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby a modified new data table is generated;

(c) generating said difference result utilizing directly or indirectly at least said modified old data table and modified new data table.

Still further, the invention provides for a system for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of :

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) scanning the old data table and for substantially each reference entry perform steps that include:

(i) replacing the reference of said entry by a distinct label mark, whereby the modified old data table is generated;

(c) reconstituting the modified new data table utilizing at least said compact difference result and said modified old data table; said modified new data table is differed from said new data table at least in that substantially each reference entry in said new data table is replaced in said modified new data table by a distinct label mark;

(d) reconstituting said new data table utilizing directly or

indirectly at least said compact difference result and said modified new data table.

Moreover, the invention provides for a system for generating a compact difference result between an old data table and a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of :

(a) generating a modified old data table utilizing at least said old data table;

(b) generating a modified new data table utilizing at least said new data table, said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/insert modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(c) generating said compact difference result utilizing at least said modified new data table and modified old data table.

The invention provides for a system for performing an update in an old data table so as to generate a new data table; each data table including reference entries that contain reference that refer to other entries in the data table; the system comprising a processing device capable of :

(a) receiving data that includes a compact difference result; said compact difference result was generated utilizing a modified old data table and a modified new data table;

(b) generating a modified old data table utilizing at least said old data table;

(c) reconstituting a modified new data table utilizing directly or indirectly at least said modified old data table and said compact difference result; said modified old data table and modified new data table have at least the following characteristics:

(i) substantially each reference in an entry in said old data table that is different than corresponding entry in said new data table due to delete/inset modifications that form part of the transition between said old data table and new data table are reflected as invariant references in the corresponding entries in said modified old and modified new data tables;

(d) reconstituting said new data table utilizing directly or indirectly at least said compact difference result and said modified new data table.

BRIEF DESCRIPTION OF THE DRAWINGS

In order to understand the invention and to see how it may be carried out in practice, a preferred embodiment will now be described, by way of non-limiting example only, with reference to the accompanying drawings, in which:

Fig. 1 is a schematic illustration of a sequence of operation according to one embodiment of the invention;

Fig. 2 shows exemplary old and new programs and the various interim results that are obtained by applying the sequence of operation of Fig 1; and

Fig. 3 is an exemplary data table, presented in the form of graph which is subject to difference extraction in accordance with the invention.

DETAILED DESCRIPTION OF A PREFERRED EMBODIMENT

In Fig. 1, module **(10)** represents the sequence of operations performed e.g. at the provider site, for generating the difference result according to one embodiment of the invention. The sequence of operation will now be described with reference to exemplary old and new programs (Fig. 2). It should be noted that the specified sequence of operations may be carried out at any known *per se* platform including, but not limited to, a conventional P.C., a computer network, etc., all as known *per se*.

Thus, P_1 stands for the old program that includes entries **(41)**, **(42)**, **(43)**, **(44)**, **(45)** and **(46)** that contain references to entries 5,8,1,1,13, and 11 respectively, (as indicated by arrows **(41')** to **(46')**.

P_2 stands for the new program which was generated (or could have been generated) by the sequence of modifications as depicted in the imaginary memory table **(80)**. Said sequence of modifications (either real or imaginary), constitutes a transition sequence between P_1 and P_2.

As shown in entry no. 6, the reference to address *'1'* was replaced (*'R'* stands for replaced) by reference to address *'11'*, e.g. due to a patch introduced by the programmer. Following the 6th entry, 3 new entries were inserted ('I' stands for inserted). The newly inserted entries reside at addresses 7 to 9. The next three entries which originally resided in addresses 7 and 9 at P_1 are now shifted (due to the 3 inserted entries) to addresses 10 to 12 in P_2.

Next, entries 10 and 11 in P_1 were deleted (D), and are therefore not assigned with any address at P_2. Entries 12 and 13 in P_1 remain intact and reside in addresses 13 and 14 in P_2. Entry 14 in P_1 was deleted (D) and it is marked as such (with no address) in the memory table **(80)**. Lastly, entry **(15)** remains intact and therefore resides in entry 15 in P_2.

Having reviewed the sequence of modifications that constitute the transition from P_1 to P_2, there follows a brief review on how the specified modifications affected the reference entries of P_1 and P_2.

Thus, and as expected, the reference 5 in entry 2 remains intact and it will therefore not appear in the difference result between P_1 and P_2.

The reference 8 in entry 4 of P_1 is now modified by reference 11 in entry 4 of P_2. The modification in the reference (from 8 to 11) in entry 4 is caused by the insertion of entries 7 to 9 in P_2, which obviously shifted entry 8 (in P_1) to entry 11 (in P_2). Before proceeding any further, it should be noted that applying conventional file difference application to P_1 and P_2, will obviously reflect that entry 4 has been changed since the reference 8 has been changed to 11. Those versed in the art will readily appreciate that according to the invention, it is desired to neutralize this change, since it has occurred solely due to the fact that other entries have been affected (i.e. entries 7 to 9). It is accordingly an object of the invention to give rise to a situation where modifications of this kind will be modified to invariant references with the obvious consequence that they are not reflected in the difference result, thereby keeping the latter relatively compact.

Reverting now to the example of Fig. 2, the next reference 1 resides in entry 6 of P_1. As recalled, this entry was intentionally modified to 11, and as expected in entry 6 of P_2, contains reference 11. Unlike the previous reference modification which stems from shifts in the program and which therefore should be neutralized from appearing in the difference results, the current modification is applied to the entry under question (i.e. the reference in entry 6 has been changed from 1 to 11) and should be reflected in the difference result. Turning now to entry 7 of P_2, it forms part of the inserted entries, and therefore reference 9 thereof is obviously not reflected in P_1. Of course, since entry 7 has been inserted, it is expected to appear in the difference results.

Entry 10, with its associated reference 13, has been deleted from P_1, and as expected, it does not appear in P_2 and should, of course, be indicated in the difference results as an entry for deletion.

Entry 9 with its associated reference 1 in P_1, corresponds to entry 12 in P_2. Since the reference in entry 12 remains 1, it will not appear in the difference result.

Turning to the last reference entry 14 in P_1, it does not appear in P_2 (since it was deleted), and therefore it is expected to appear in the difference result as an entry for deletion.

Having described in general the differences between P_1 and P_2, as well as their effect on the difference result, attention is now directed also to Fig. 1 for describing one non-limiting realization of a system and method of the invention for accomplishing the desired difference result.

By this particular embodiment, the desired invariant references are accomplished by generating modified old and new programs wherein address references in entries are replaced by label marks as follows:

a) Create P''_1 table from P_1 (steps **(201)** and **(202)** in Fig. 1) and P''_2 table from P_2 (steps **(203)** and **(204)** in Fig. 1) by adding label marks and replacing references in entries with some fixed values. As shown in Fig. 2, P''_1 is generated from P_1 by adding label marks to entries 1, 5, 8, 11 and 13 (designated **(101)** to **(105)**, respectively). As also shown in Fig. 2, the references at entries 2, 4, 6, 9 10 and 14 were set to a fixed value and by this particular example 0. Although not shown in Fig 2, P''_2 is generated in a similar manner;

b) Create a translation table L_1 (step **(205)** in Fig. 1) between entry references in P_1 and label distinct values. Thus, and as shown in Fig. 1 **(110)**, a distinct label number is assigned to each label mark of P''_1. By this particular example, the distinct labels are assigned in ascending order and, as shown, labels marks **(101)** to **(105)** are assigned with the respective values 1 to 5;

c) P''_1 and P''_2 are compared giving rise to difference table D_1 **(206)** using file difference utilities of the kind specified above. As shown in Fig. 2, D_1 **(120)** contains list of entries each having the structure of

<X, *n*>, where X stands for C (copy), I (insert), D(delete), or T (Toggle) and *n* stands for the number of instructions. D_1 includes, in fact, a list of instructions for generating P''_2 from P''_1. By this particular example, D_1 includes the following entries: <C,6> which signifies that the first 6 entries of P''_1 should be copied to P''_2. Note that whilst the 6th entry is different in P_1 and P_2 (i.e. the reference has been changed from 1 to 11), this entry is the same in P''_1 and P''_2 ,since both references were set to 0 and, accordingly, the 6th entry forms part of the copied part. The next entry <I,3> ,signifies that three new entries should be inserted, and this accounts for the new three entries 7 to 9 that were inserted to P_2 (and are also reflected in P''_2). The next entry <C,3> stands for the three entries that reside in addresses 7 to 9 in P''_1 (and P_1), and are shifted (without affecting their contents), to addresses 10 to 12 in P''_2 (and P_2). The next entry <D, 2> stands for the two entries (10 and 11) that were deleted from P_1 (and obviously also from P''_1). The next entry <C,1> stands for the entry 12 in P''_1 that was shifted (without affecting its contents) to entry 13 in P''_2. The next entry <T,1> is a special entry indicating that a change has occurred in the label of entry 13 of P''_1, i.e,. it was removed in the corresponding entry 14 in P''_2 for the simple reason that the entry that refers to this label was deleted (entry 10 in P''_1 (and P_1). <T,1> signifies, thus, that the entry 13 should be copied (to entry 14), whilst deleting the label mark. Next, <D,1> stands for the deleted entry (14) in P''_1, and <C,1> stands for copying the last intact entry that is to be copied from P''_1 to P''_2 (entry 15).

d) Analyzing D_1 to determine the *P*osition and size of deleted or inserted program fragments and apply equivalent changes to L_1 translation table to create L_2 translation table **(207)**, which translates entry

references in P_2 to their distinct label value in the following manner: For a non-inserted referred entry perform:

- L_2[entry address] = L_1[matching entry address in P_1] (hereinafter the *first condition*)

For an inserted referred entry, perform:

- L_2[entry address] = U( ) different than any value in L_1; U( ) signifies a label generation function for generating distinct labels (hereinafter the *second condition*). The function is of repeatable nature, namely, when activated in the same scenario it will always generate the same result. The latter characteristics will be clarified when the subsequent program reconstruction phase is described below.

The resulting L_2 table **(130)** is shown in Fig.2. Thus, the first six instructions fall in the first condition (i.e. non inserted entry) and, accordingly, the distinct labels (1 and 2) are simply copied from L_1.

The next three entries are inserted and, accordingly, the second condition applies and U( ) is activated to generate a distinct label value. Since, as shown in L_1, five labels are "occupied", the next free one is 6 and, accordingly, the reference entry 9 is assigned with the value 6. Those versed in the art will readily appreciate that whilst in this example U( ) generates the distinct label values by simply incrementing the last occupied value by 1, this is only an example.

Turning now to the next three entries, <C,3> , the reference entry 11 falls in the first criterion and, accordingly, the label value (3) is taken from L_1.

The next entry in D_1 <D,2> is ignored, since it concerns two deleted entries.

The next entry, <C,1>, has no reference entry and therefore need not be processed for generating L_2.

The next entry, <T,1> ,corresponds to the deleted label (from entry 13 in L_1) and in this respect, is resembles the previous delete modification that is ignored.

The last two entries, <D,1> and <C,1>, do not involve reference entries and therefore need not be processed for the purpose of generating L_2.

e) Create P'_1 **(208)** and P'_2 **(209)** for P_1 and P_2 respectively, by replacing entry references with their translated values using L_1 and L_2 tables and by coping label marks from P''_1 and P''_2.

Step (e) will be described with reference to P'_1 and applies *mutatis mutandis* also to P'_2 (see **150** in Fig. 2). Thus, the label marks **(101)** to **(105)** of P''_1 are copied to the respective locations in P'_1 (**140** in Fig. 2). Next, the reference entries of P_1 are replaced by their corresponding label marks as retrieved from L_1. More specifically, the reference 5 in entry 2 of P_1 is replaced by the corresponding label from L_1. As shown, label 2 resides in entry 5 of L_1 and, accordingly, the reference in entry 2 in P'_1 is set to 2.

In a similar manner, the reference 8 in entry 4 of P_1 is replaced by label number 3 according to the label number (3) that resides in entry 8 of L_1. In a similar manner, the references 1,1,13 and 11 of entries 6,9,10 and 14 of P_1 are replaced by respective label numbers 1,1,5 and 4.

f) Having generated P'_1 and P'_2, the final difference result D_2 is generated. To this end, D_1 is analyzed to determine the position of program fragments copied from P_1 to P_2 (i.e. in the example of Fig. 2, the entries that fall in <C, x> or <T,x> commands; the <I, x> <D,x> commands are ignored>). For copied entries as derived from D_1, P'_1 and P'_2 are compared so as to generate D_2 **(210)** in the following manner:

f1) take each pair of matching entries in P'_1 and P'_2 (neither deleted nor inserted) that contain a (replaced) reference, and compare their replaced reference values. In the case of discrepancy, add a special modification command to reflect the difference.

f2) attach all the inserted program fragments and replaced values. These fragments are taken from P'_2, thus they contain label marks, and address references remain under L_2 translation.

Step (f) will now be exemplified with reference to the specific example of Fig. 2. As recalled, only entries that fall in a <C,x> command are of interest for the f1 step analysis. Thus, the first non-inserted entry of D_1, i.e. <C,6> is analyzed. According to step f1, only the sixth entry contains a replaced reference (reference 1 in P'_1 as compared to reference 3 in P'_2). Accordingly, The first command of D_1 <C,6> is replaced in D_2 by <C,5> and a correction command <R,1> standing for "replace label in entry 6" is added to D_2. Entries 2 and 4 in P'_1 (that also fall in the first six entries and are encompassed by the <C,6> command), contain the respective references 2 and 3, exactly as the corresponding entries in P'_2 and, accordingly, no correction command is required.

The rest of the entries that correspond to <C,3> <C,1> commands do not include replaced references in P'_1 and P'_2 and, accordingly, no replacement command is required.

Step f2 simply stipulates that the inserted data and replaced data (replaced reference) will be appended to D_2 in order for the reconstructing party to be able to reconstitute P_2 from D_2 and P_1.

Thus the three entries (that should be inserted to entries 7 to 9 in P_2) and which correspond to <I,3> in D_2, are added to D_2 in section **(161)**. Likewise, the replaced reference 3 (instead of 1) in the sixth entry which corresponds to <R,1> in D_2 is added to D_2 ,in section **(161)**.

Depending upon the particular application, D_2 may be stored on a storage medium, or transmitted through a communication network (212 in Fig. 2), all as required and appropriate.

There follows a description for a typical sequence of operations (220) for reconstructing P_2 from P_1 and the so received D_2, according to the present embodiment. Reconstructing P_2 may also be realized on any desired platform, all as required and appropriate. A sequence of operation, according to this embodiment includes:

a) Generate L_1 (222) from P_1 (221); (see step b above in 200)

b) Generate P'_1 (223) from P_1 and L_1; (see step e above in 200)

c) Generate P'_2 (224) from P_1' and D_2 (210) by applying the modification commands of D_2 on P'_1;

d) Analyze the so received D_2 difference result to determine the position and size of copied program fragments of P_1 and use L_1 to create AL_2 (225 in Fig. 1), which translates the label enumeration values appearing in P'_2, back to their original address reference in the following manner:

For a non-inserted referred address:

$AL_2[L_1[\text{matching address in } P'_1]]$ = address (first condition)

For an inserted referred address:

$AL_2[U(\,)]$ = address (second condition)

Step (d) will now be exemplified with reference to the specific example of Fig. 2. Thus, in order to generate AL_2 (170), at first L_2 (130) is reconstructed, using to this end L_1, P'_2 and D_2 all of which are available at the reconstruction side (220). Having reconstructed L_2, AL_2 can be easily derived by reversing L_2. More specifically, entry 1 in L_2 holds the value 1 and accordingly, entry 1 of AL_2 holds the value 1. Entry 5 of L_2 holds the value 5 and accordingly, entry 2 in

AL_2 holds the value 5. By following the same reverse logic, entry 9 of L_2 holds the value 6 and accordingly entry 6 of AL_2 holds the value 9 and lastly by following the same logic entry 3 of AL_2 holds the value 11.

e) Lastly, P_2 **(226)** is reconstructed from P'_2 **(224)** and AL_2 **(225)** by translating address references in P'_2 from label enumeration values back to the original address references using AL_2.

Step (e) will now be exemplified with reference to the specific example of Fig. 2. Thus, in order to reconstruct P_2 **(180)**, the label references in P'_2 are translated according to AL_2. More specifically, label reference 2 in entry 2 of P'_2 is replaced by actual entry reference 5 in P_2 according to the value 5 in entry # 2 of AL_2. Likewise, label references 3 in entries 4 and 6 of P'_2, are replaced by actual address reference 11 in P_2, according to the value 11 in entry # 3 of AL_2. In a similar manner, label references 1 in entry 12 of P'_2 is replaced by actual address reference 1 in P_2 according to the value 1 in entry # 1 of AL_2. Lastly, label reference 6 in entry 7 of P'_2, is replaced by actual address reference 9 in P_2, according to the value 9 in entry # 6 of AL_2. As shown by this particular example, P_2 was generated indirectly from the difference result data D_2, by the intermediary data structure, AL_2.

Entries 4 and 5 of AL_2 hold the value 0, signifying that reference labels 4 and 5 need not be updated in P_2, since they form part of entries in P_1 that were deleted.

Those versed in the art will readily appreciate that whilst the invention has been described with reference to a specific application of updating software through a communication network, the invention is by no means bound by this specific application. Thus, by way of another example, the invention is applicable

for efficient version control. Consider, for example, a series of program versions P_1 P_2 and P_3. Applying the technique of the invention gives rise to compact D_{12} and D_{23}, which stand for the difference between P_1 P_2 and P_2 P_3, respectively. The resulting compact D_{12} and D_{23} ,as compared to conventional, larger in size, difference results, bring about the desired efficient version control tool. Other applications are, of course, feasible, all as required and appropriate.

Having described the invention with reference to a specific application of extraction differences between old and new versions of computer programs, there follows a description with reference to a more generalized representation of data table depicted in the form of a graph.

Before turning to the specific example, few observations which apply to the data table representation are set forth below:

- The graph represents a collection of data-records inter-connected by references which enable to access them.
- Data records are stored in a linear storage such as array according to a pre-defined order.
- The amount of storage allocated for storing the references is not negligible.
- The data that represents the references is not held as a contiguous block, but rather is dispersed among other items of the data table (e.g. among the data records).

Bearing this in mind, attention is directed to Fig. 3A. The graph (**300**) represents an (old version of an) abstract data structure in the form of a graph (which is applicable in many applications as will be explained in greater details below). The nodes A-E represent data-records and the links represent references to other data-records. In this example, the pre-defined order for the data records is an alphabetical sort of records' names. The storage of the graph would appear as:

1. A, 2, 4
2. B, 3
3. C, 4, 5
4. D
5. E, 2

Thus, for example, data-record A (**301**), having address 1, is linked by means of links (**302** and **303**) that stand for references 2 and 4 respectively, to data-records B ad D (**304** and **305**). As shown above, B and D reside in addresses 2 and 4 respectively.

Should it now be required to modify the above graph by adding a data-record named ABA (**306** in Fig. 3B) that is linked by means of link (**307**) to data-record D (**305**), this would bring about the following new graph storage, where ABA is inserted in accordance with the said alphabetical order:

1. A , 3, 5
2. ABA, 5
3. B, 4
4. C, 5, 6
5. D
6. E, 3

It is accordingly appreciated that insertion of only one data-record gave rise to fairly large differences between the storage of the old and the storage of the new graphs.

It is readily appreciated that the above abstract data structure is similar to a computer program, except for the fact that in the specified data structure, alphabetical order is imposed as compared to a computer program, where the order of execution is imposed.

Accordingly, applying the technique described in detail with reference to Figs. 1 and 2 above would give rise to an extraction of compact differences.

As specified above, the data table is by no means bound to the representation of a computer program. Thus, by way of non-limiting example, the specified graph (**300**) represents a map where nodes stand for cities and links for roads linking the various cities. The technique of the invention would allow for compact representation of topographical modification in the map (e.g. in response to the construction of a new road between the City having the symbol **ABA** and the city having the symbol **D**).

By way of another non-limiting example, the specified graph represents electronic circuitry where the nodes stand for the components and the links stand for the wiring connections between the components.

Numbers, alphabetic characters and roman symbols that appear in the following claims are designated for convenience of explanations only, and do not necessarily imply the particular order of the steps.

The invention has been described with a certain degree of particularity, but those versed in the art will readily appreciate that various alterations and modifications may be carried out without departing from the spirit and scope of the following Claims: